Apple Assembly Line

$1.80

Volume 5 -- Issue 10 July, 1985

In This Issue...

ProDOS Macro Assembler

We are now shipping the ProDOS version of the S-C Macro Assembler. As reported last month, the ProDOS version alone is $100 and the DOS and ProDOS versions together are $120. The ProDOS update for owners of the DOS Version 2.0 is $30, and for owners of DOS Version 1.x is $50.

The S-C Cross Reference Utility and the Laumer Research Full Screen Editor have been updated to ProDOS versions. The ProDOS code will be included on the back of the disk in all new shipments, and current owners can return their original disks to be updated at a cost of only $5 per program.

65802 Chips

Good News! We have arranged a quantity price on 65802 processors, so we will be able to sell them to our readers for only $50 + shipping. That's only $51.50 in the US for this powerful new 16-bit processor that plugs right into your Apple II, II+, //e, or //c. Combine this chip with the S-C Macro Assembler Version 2.0 and you can start writing faster, more compact code. Order yours today!

Updated VideoTerm Driver

We recently revised the Videx VideoTerm driver in the S-C Macro Assembler Version 2.0 to make it firmware-independent and ViewMaster-compatible. This revision is effective with Serial Number T-1483, so owners of earlier copies can send in their original disks and $5 for an updated copy.

Reading DOS 3.3 Disks With ProDOS.....Bob Sander-Cederlof

At the track and sector level, DOS 3.3 disks are identical to ProDOS disks. They both have 35 tracks, 16 sectors, and the sectors are laid out on the tracks the same way in both systems. You can use DOS's COPYA program to copy ProDOS disks, and you can use some ProDOS utilities on DOS disks.

The structure of the files is of course entirely different between the two systems. Hence the need for the CONVERT program found on ProDOS system master disks, and the System Utilities Disk that comes with the //c. Unfortunately both of the above programs have bugs that get in the way nearly every time I want to move a file from DOS to ProDOS. The one that bites me the most is the way CONVERT dies when it encounters a DOS filename which does not start with a letter. We routinely use such "illegal" filenames on our disks to separate and identify sections of long catalogs, but CONVERT goes absolutely crazy when it finds one.

Therefore, I decided to write a program which could "LOAD" assembler source files from a DOS 3.3 disk while I am running the ProDOS version of the S-C Macro Assembler. Even with error messages and other fancy features, the program turns out to be only a little over $280 bytes long, and it works.

It is based on the fact that the Block Read MLI call does not care whether the disk being read is a DOS or a ProDOS disk. The Block Read MLI call reads 512 bytes, or two sectors, at a time. The call looks like this:

```
       JSR $BF00        (MLI link in global page)
       .DA #$80         (block read code)
       .DA PARMLIST     (address of parameters)
```

MLI returns with carry clear if there was no error, or carry set if there was an error. The error code will be in the A-register if there was an error.

The PARMLIST for Block Read looks like this:

```
    PARMLIST .DA #3           (3 parameters)
             .DA #$60         (1-byte unit number)
             .DA BUFFER       (address of 512-byte buffer)
             .DA 2            (2-byte block number)
```

Page 3-17 of "Beneath Apple ProDOS" contains a table which converts block numbers to physical track/sector, and vice versa. The latest printing of the book also includes a line which correlates the physical sector values to the DOS 3.3 logical sector. Boiling it down, you can derive a ProDOS block number from the DOS 3.3 logical sector by multiplying the track number by 8 and adding a value according to the sector number from the following table:

```
    DOS sector #:  0 1 2 3 4 5 6 7 8 9 A B C D E F
                   0 7 6 6 5 5 4 4 3 3 2 2 1 1 0 F
```

S-C Macro Assembler Version 2.0 (DOS or ProDOS).....$100
S-C Macro Assembler Version 2.0 (DOS and ProDOS).....$120
ProDOS Upgrade Kit for Version 2.0 DOS owners.....$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.....$20
Source Code for Version 1.1 (on two disk sides).....$100
Full Screen Editor for S-C Macro (with complete source code).....$49
S-C Cross Reference Utility (without source code).....$20
S-C Cross Reference Utility (with complete source code).....$50
DISASM Dis-Assembler (RAK-Ware).....$30
Source Code for DISASM.....additional $30
S-C Word Processor (with complete source code).....$50
DP18 Source and Object.....$50
Double Precision Floating Point for Applesoft (with source code).....$50
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk.....$15

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.....each $15, or any four for $45

		Jan-Mar	Apr-Jun	Jul-Sep	Oct-Dec
Each disk contains	1980	-	-	-	1
the source code from	1981	2	3	4	5
three issues of AAL,	1982	6	7	8	9
saving you lots of	1983	10	11	12	13
typing and testing.	1984	14	15	16	17
	1985	18	19		

AWIIe Toolkit (Don Lancaster, Synergetics).....$39
ES-CAPE: Extended S-C Applesoft Program Editor (new price, was $60) $40
"Bag of Tricks", Worth & Lechner, with diskette.....($39.95) $36
MacASM -- Macro Assembler for MacIntosh (Mainstay).....($150.00) $100

Blank Diskettes (Verbatim)..... package of 20 for $32
(Premium quality, single-sided, double density, with hub rings)
Vinyl disk pages, 6"x8.5", hold two disks each.....10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).....40 cents each
(Cardboard folders designed to fit 6"X9" Envelopes.) or $25 per 100
Envelopes for Diskette Mailers..... 6 cents each

65802 Microprocessor (Western Design Center).....($95) $50
quikLoader EPROM System (SCRG).....($179) $170
PROmGRAMER (SCRG).....($149.50) $140
Switch-a-Slot (SCRG).....($190) $175
Extend-a-Slot (SCRG).....($35) $32

"Apple ProDOS: Advanced Features for programmers", Little..($17.95) $17
"Inside the Apple //c", Little.....($19.95) $18
"Inside the Apple //e", Little.....($19.95) $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95) $18
"Apple][Circuit Description", Gayler.....($22.95) $21
"Understanding the Apple II", Sather.....($22.95) $21
"Understanding the Apple //e", Sather.....($24.95) $23
"Enhancing Your Apple II, vol. 1", Lancaster.....($15.95) $15
"Enhancing Your Apple II, vol. 2", Lancaster.....($17.95) $17
"Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95) $20
"Beneath Apple DOS", Worth & Lechner.....($19.95) $18
"Beneath Apple ProDOS", Worth & Lechner.....($19.95) $18
"6502 Assembly Language Programming", Leventhal.....($18.95) $18
"6502 Subroutines", Leventhal.....($18.95) $18
"Real Time Programming -- Neglected Topics", Foster.....($9.95) $9
"Microcomputer Graphics", Myers.....($12.95) $12
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95) $16
"Assembly Lines -- the Book", Wagner.....($19.95) $18
"AppleVisions", Bishop & Grossberger.....($39.95) $36

Add $1.50 per book for US shipping. Foreign orders add postage needed.
Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We accept Master Card, VISA and American Express ***

For example, track 0 sector 2 is in ProDOS block 6. The only problem is, so is DOS track 0 sector 3. We also need to remember whether a given sector is in the upper or lower half of a 512-byte block.

I developed the following subroutine, which will translate the DOS logical track and sector numbers into the appropriate block number, read the block, and return with the address of the buffer page in which the sector data has been read. Call the routine with the track number in the A-register and the sector number in the X-register. The high-byte of the buffer address will return in the X-register. If MLI detects an error, the subroutine will return with carry set.

```
RTS     LDY #0  ASSUME BLOCK # < $100
        ASL     FORM TRACK*8
        ASL
        ASL
        BCC .1  ...BLOCK < $100
        INY     ...BLOCK > $0FF
.1      ASL     *2, MAKE ROOM FOR H/L FLAG BIT
        ORA BLKTBL,X   MERGE FROM SECTOR TRANSLATION
        ROR     H/L FLAG BIT TO CARRY
        STA BLOCK
        STY BLOCK+1
        LDX /BLOCK.BUFFER   HIGH BYTE OF BUFFER ADDRESS
        BCC .2  ...LOWER HALF OF BUFFER
        INX     ...UPPER HALF OF BUFFER
.2      JSR $BF00
        .DA #$80,PARMLIST
        RTS

BLKTBL .HS 00.0E.0D.0C.0B.0A.09.08
       .HS 07.06.05.04.03.02.01.0F

PARMLIST
       .DA #3
       .DA #$60         SLOT 6, DRIVE 1
       .DA BLOCK.BUFFER
BLOCK  .DA 0            <FILLED IN>
```

After playing with the subroutine a while, I proceeded to write the load program. Using a well-worn copy of "Beneath Apple DOS", I figured out once more how to work through a DOS catalog. I decided to display a menu of files on the screen, and allow a single keystroke to select a file to be loaded.

The program that follows is designed to work with the ProDOS version of the S-C Macro Assembler. Assuming it has been assembled and is in a ProDOS binary file as DOS.LOAD, and assuming you have booted the ProDOS version of the S-C Macro Assembler, you can start up the load program by typing "-DOS.LOAD". It will load source files from DOS disks, which are DOS type I files, and place them in the assembler's edit area. After selecting the slot and drive, the program reads the DOS catalog and displays 20 filenames at a time. Only type I filenames are displayed, any others are skipped over. If there are more than 20 files, you can page through them. If

you change your mind about loading a file, you can abort. If you see the file you want to load, you type a single letter to select it. A few seconds later it has been loaded, and you are returned to the assembler.

The assembler's soft entry point is at $8003, and the load program jumps there after finishing a load or after encountering an error. Three pointer locations in page zero which the assembler uses are used by the load program: HIMEM ($73,74) points one byte higher than the program can be loaded; PP ($CA,CB) will point to the beginning of the program, if it is successfully loaded; LOMEM ($67,68) points to the lowest address the program can occupy. HIMEM is normally at $7400, and LOMEM at $1000, but these can be changed with the HIMEM and LOMEM commands. LOMEM could be set as low as $0800.

With these limitations on the program extent ($0800...73FF), you can see that the maximum size assembler source file that can be loaded from a DOS disk is $6C00 bytes, or 108 sectors. Or, if you prefer to leave LOMEM at $1000, you can load $6400 bytes or 100 sectors. Most likely you do not have any source files which are bigger than that anyway. If you do, you need to load the DOS version of the assembler and split the files before they can be transferred to ProDOS. The maximum size file of 108 data sectors would only have one track/sector list, so I did not include any logic to chain to a second track/sector list. You may be wondering where the load program itself loads....

The command interpreter I developed for the ProDOS version of the S-C Macro Assembler has three 1024-byte buffers permanently allocated between $7400 and $7FFF. None of them will be in use while the load program is executing, so I borrowed some of that space for the load program. The load program itself loads inside the buffer space allocated to the EXEC command, at $7400-77FF. The blocks read by MLI will be stored at $7C00-7DFF, and I will save a copy of the track/sector list for the file being loaded at $7E00-7EFF.

Now for a description of the actual code. Lines 1270-1410 ask you to type in the slot and drive numbers of the floppy drive the DOS disk is in. ProDOS uses a "unit number", which is a coded form of the slot and drive all in one byte. The slot number is in bits 4-6 and the drive number (0 or 1, corresponding to drives 1 or 2 respectively) in bit 7. My subroutine GETNUM prints a prompt message (selected by the Y-register), inputs a single character from the keyboard, and checks it for legal range. GETNUM is designed to accept only digits, starting with "1", and up to but not including the value in the A-register when GETNUM is called.

Once the unit number has been established, we fall into the LOAD.MENU code. This code is somewhat convoluted, enough to disgust even me. Interlocking loops? Multiple entries and exits? Ouch! Maybe it really IS structured code, but just not in Euclidean space. I think maybe it could be diagrammed on the surface of a Klein bottle (recursive torus?).

Anyway, let's walk through it. Line 1440-1500 set up a fresh menu display and read in the DOS VTOC page so we can start reading the catalog. The second and third bytes in the VTOC page give the track and sector of the first catalog sector. This is almost always track $11, sector $0F; however, by starting at VTOC, we are a little more general. We are still assuming we know where the VTOC is, which is track $11, sector 0. Some non-standard software sets up disks with the VTOC somewhere else, but you are very unlikely to find any S-C source code on such a disk. Each sector of the catalog also contains the track/sector of the next catalog sector in the 2nd and 3rd bytes.

Lines 1530-1550 read in the next catalog sector and set the pointer to the first file entry in that sector. Each file entry is 35 bytes long, and the first one starts at $0B within the sector. The subroutine READ.NEXT.CATALOG.SECTOR will return with carry set if there are no more catalog sectors. The first time through this code, when we fall in from the code above, we will read the first catalog sector.

Lines 1570-1960 pick up filenames out of the catalog sectors and write them on the screen. Not all file names are used: line 1610 filters out deleted files; lines 1660-1700 filter out files which are not type I. The track and sector of the active type-I files are saved in an array, indexed by the menu letter. These values are first picked up in lines 1620-1650, and added to the array in lines 1870-1940. Lines 1720-1770 print the menu letter and two dashes, and then lines 1780-1850 print the filename.

Lines 1950-1960 decrement the line count and test if the screen is full yet. I arbitrarily call a screen full if it has 20 filenames, leaving room for my three-line prompt message. We jump to MENU.SELECTION when we reach 20 lines or when we reach the end of the catalog, whichever comes first

If we are not yet at the end of catalog and have not yet filled the screen, or if the file was one that got filtered out of the menu, we come to GET.NEXT.FILE at line 1980. Lines 1990-2040 update the pointer into the catalog sector so that it points at the next file, if there is another one. If so, we branch back to NEXT.FILE.NAME, to try the next one in the current sector. If no more names in this sector, we go back to NEXT.CAT.SECTOR to get the next catalog sector (if any).

When we reach the end of catalog, lines 2070,2080 set a flag. We need a flag to tell whether it was screen-full or catalog-end which caused us to come to MENU.SELECTION, so we can either continue through the catalog or wrap-around to the beginning should you wish to see another screenful of filenames.

The MENU.SELECTION section prints a three-line prompt message and waits for you to type a character. If you type a space, you seethe next screenful of filenames. (Of course, if there are fewer than 21 type I files on the disk you will see the same ones over again.) If you type the RETURN or ESCAPE keys, the load program will abort, returning directly to the

assembler without loading a file. If you type a letter in the range of the menu, that file will be loaded. Any other key is ignored.

Lines 2260-2370 convert the menu letter you typed into an index to get the track and sector for the track/sector list of the selected file. The track/sector list contains the track and sector for every data sector in the file. Line 2310 reads the track/sector list, and lines 2330-2370 copy it into a special buffer.

The first two bytes of the first data sector of a type-I file contain the length of the file. We need to know the length so we can figure out where to read the data. Lines 2390-2510 read in the first data sector and get the file size.

Lines 2520-2630 figure out where PP should be set so that the file exactly fits between PP and HIMEM, and checks to make sure that it does not go below LOMEM.

Lines 2650-2670 copy the rest of that first sector into the load area, starting at PP. If the file is so short it doesn't fill the first data sector, the LOAD.FROM.SECTOR subroutine will return with carry set and we will return to the assembler, all finished. Otherwise, we fall into the code below, to load the succeeding data sectors. Eventually we will bump into HIMEM, and we are finished.

Now that this program is working I can see neat ways to extend it. Why restrict it to type-I files? It could also BLOAD type-B files, as long as an appropiate load address was set up. It could do the equivalent of a BLOAD on a type-T file, which then could be BSAVE as type TXT in ProDOS. Seems like we might be able to do away with the need for CONVERT, at least in the direction of moving from DOS to ProDOS.

```
7400-           1030         .TF DOS.LOAD
                1000 *SAVE S.DOS.LOAD
                1010 *--------------------------------
                1020         .OR $7400
7400-           1030         .TF DOS.LOAD
                1040 *--------------------------------
00-             1050 PNTR            .EQ $00.01
02-             1060 CAT.INDEX       .EQ $02
03-             1070 MENU.LETTER     .EQ $03
04-             1080 LINE.COUNT      .EQ $04
05-             1090 TRACK           .EQ $05
06-             1100 SECTOR          .EQ $06
07-             1110 DONE.FLAG       .EQ $07
08-             1120 SIZE            .EQ $08,09
0A-             1130 LIMIT           .EQ $0A
                1140 *--------------------------------
67-             1150 LOMEM           .EQ $67,68
73-             1160 HIMEM           .EQ $73,74
CA-             1170 PP              .EQ $CA,CB
                1180 *--------------------------------
7C00-           1190 BLOCK.BUFFER    .EQ $7C00
7E00-           1200 TS.LIST         .EQ $7E00
                1210 *--------------------------------
FD0C-           1220 MON.RDKEY       .EQ $FD0C
FD8E-           1230 MON.CROUT       .EQ $FD8E
FDDA-           1240 MON.PRHEX       .EQ $FDDA
FDED-           1250 MON.COUT        .EQ $FDED
                1260 *--------------------------------
```

```
                    1270 DOS.LOAD
7400- A0 76         1280         LDY #EM3      "SLOT:"
7402- A9 B8         1290         LDA #"8"      1...7
7404- 20 3C 75      1300         JSR GETNUM    00000SSS
7407- 4A            1310         LSR           000000SS S
7408- 6A            1320         ROR           S000000S S
7409- 6A            1330         ROR           SS000000 S
740A- 6A            1340         ROR           SSS00000
740B- 8D 64 76      1350         STA UNIT
740E- A0 7E         1360         LDY #EM4      "DRIVE:"
7410- A9 B3         1370         LDA #"3"      1...2
7412- 20 3C 75      1380         JSR GETNUM
7415- 4A            1390         LSR
7416- 4A            1400         LSR
7417- 6E 64 76      1410         ROR UNIT      DSSS0000
                    1420 *--------------------------------
                    1430 LOAD.MENU
741A- 20 96 75      1440         JSR SETUP.SCREEN
741D- A9 11         1450         LDA #17             TRACK 17
741F- A2 00         1460         LDX #0              SECTOR 0
7421- 86 07         1470         STX DONE.FLAG
7423- 86 00         1480         STX PNTR
7425- 20 68 75      1490         JSR RTS             READ DOS 3.3 VTOC
7428- 86 01         1500         STX PNTR+1          SET POINTER
                    1510 *--------------------------------
                    1520 NEXT.CAT.SECTOR
742A- 20 52 75      1530         JSR READ.NEXT.CATALOG.SECTOR
742D- B0 56         1540         BCS END.OF.CATALOG
742F- A0 0B         1550         LDY #$0B
                    1560 *--------------------------------
                    1570 NEXT.FILE.NAME
7431- 84 02         1580         STY CAT.INDEX
7433- B1 00         1590         LDA (PNTR),Y        TRACK
7435- F0 4E         1600         BEQ END.OF.CATALOG
7437- 30 42         1610         BMI GET.NEXT.FILE ...DELETED FILE
7439- 85 05         1620         STA TRACK
743B- C8            1630         INY
743C- B1 00         1640         LDA (PNTR),Y
743E- 85 06         1650         STA SECTOR
7440- C8            1660         INY
7441- B1 00         1670         LDA (PNTR),Y        FILE TYPE
7443- 0A            1680         ASL                 IGNORE LOCK BIT
7444- C9 02         1690         CMP #2              MUST BE TYPE I
7446- D0 33         1700         BNE GET.NEXT.FILE ...NOT I, SKIP OVER IT
                    1710 *---DISPLAY MENU LINE-------------
7448- A5 03         1720         LDA MENU.LETTER
744A- 20 ED FD      1730         JSR MON.COUT        DISPLAY MENU LETTER,
744D- E6 03         1740         INC MENU.LETTER
744F- A9 AD         1750         LDA #"-"
7451- 20 ED FD      1760         JSR MON.COUT        ...TWO DASHES
7454- 20 ED FD      1770         JSR MON.COUT
7457- A2 1E         1780         LDX #30
7459- C8            1790 .1      INY
745A- B1 00         1800         LDA (PNTR),Y
745C- 09 80         1810         ORA #$80
745E- 20 ED FD      1820         JSR MON.COUT        ...AND FILENAME
7461- CA            1830         DEX
7462- D0 F5         1840         BNE .1
7464- 20 8E FD      1850         JSR MON.CROUT
                    1860 *---SAVE T/S OF TS-LIST-----------
7467- A5 03         1870         LDA MENU.LETTER
7469- 29 1F         1880         AND #$1F            CONVERT TO INDEX
746B- AA            1890         TAX
746C- CA            1900         DEX                 ...SINCE LETTER INC'ED ALREADY
746D- A5 05         1910         LDA TRACK
746F- 9D 69 76      1920         STA TRACKS,X
7472- A5 06         1930         LDA SECTOR
7474- 9D 7E 76      1940         STA SECTORS,X
7477- C6 04         1950         DEC LINE.COUNT
7479- F0 0E         1960         BEQ MENU.SELECTION  BRANCH IF SCREEN FULL
                    1970 *--------------------------------
                    1980 GET.NEXT.FILE
747B- 18            1990         CLC
747C- A5 02         2000         LDA CAT.INDEX
747E- 69 23         2010         ADC #35
7480- A8            2020         TAY                 BUMP INDEX
7481- 90 AE         2030         BCC NEXT.FILE.NAME
7483- B0 A5         2040         BCS NEXT.CAT.SECTOR
                    2050 *--------------------------------
```

```
                        2060 END.OF.CATALOG
7485- A9 01             2070         LDA #1
7487- 85 07             2080         STA DONE.FLAG
                        2090 MENU.SELECTION
7489- A0 00             2100         LDY #EMO              3-LINE PROMPT
748B- 20 30 75          2110         JSR PRINT.MSG
748E- 20 0C FD          2120 .2      JSR MON.RDKEY
7491- C9 E0             2130         CMP #$E0              LOWER CASE?
7493- 90 02             2140         BCC .3
7495- 29 DF             2150         AND #$DF              STRIP CASE
7497- C9 A0             2160 .3      CMP #" "              SPACE?
7499- F0 7D             2170         BEQ MENU.NEXT.SCREEN
749B- C9 8D             2180         CMP #$8D              RETURN?
749D- F0 76             2190         BEQ ABORT
749F- C9 9B             2200         CMP #$9B              ESCAPE?
74A1- F0 72             2210         BEQ ABORT
74A3- C9 C1             2220         CMP #"A"
74A5- 90 E7             2230         BCC .2                NOT A-Z, SO IGNORE
74A7- C5 03             2240         CMP MENU.LETTER
74A9- B0 E3             2250         BCS .2                BEYOND VALID VALUES
                        2260 *---GET T/S LIST----------------
74AB- 29 1F             2270         AND #$1F              CONVERT LETTER TO INDEX
74AD- A8                2280         TAY
74AE- BE 7E 76          2290         LDX SECTORS,Y
74B1- B9 69 76          2300         LDA TRACKS,Y
74B4- 20 68 75          2310         JSR RTS               READ TRACK/SECTOR LIST
74B7- 86 01             2320         STX PNTR+1            SET POINTER
74B9- A0 00             2330         LDY #0
74BB- B1 00             2340 .4      LDA (PNTR),Y          MOVE T/S LIST TO ITS BUFFER
74BD- 99 00 7E          2350         STA TS.LIST,Y
74C0- C8                2360         INY
74C1- D0 F8             2370         BNE .4
                        2380 *---GET THE FILE SIZE------------
74C3- A0 0C             2390         LDY #$0C              POINT AT FIRST T/S
74C5- 84 02             2400         STY CAT.INDEX
74C7- B9 00 7E          2410         LDA TS.LIST,Y         TRACK
74CA- F0 59             2420         BEQ ERR.EMPTY.FILE
74CC- BE 01 7E          2430         LDX TS.LIST+1.Y       SECTOR
74CF- 20 68 75          2440         JSR RTS               READ FIRST SECTOR
74D2- 86 01             2450         STX PNTR+1
74D4- A0 00             2460         LDY #0
74D6- B1 00             2470         LDA (PNTR),Y          GET FILE SIZE
74D8- 85 08             2480         STA SIZE
74DA- C8                2490         INY
74DB- B1 00             2500         LDA (PNTR),Y
74DD- 85 09             2510         STA SIZE+1
                        2520 *---MAKE ROOM FOR FILE-----------
74DF- 38                2530         SEC
74E0- A5 73             2540         LDA HIMEM
74E2- E5 08             2550         SBC SIZE
74E4- 85 CA             2560         STA PP                SET ASSEMBLER'S POINTER
74E6- 8D B6 75          2570         STA LPTR+1            AND OUR LOAD POINTER
74E9- A5 74             2580         LDA HIMEM+1
74EB- E5 09             2590         SBC SIZE+1
74ED- 85 CB             2600         STA PP+1
74EF- 8D B7 75          2610         STA LPTR+2
74F2- C5 68             2620         CMP LOMEM+1
74F4- 90 32             2630         BCC ERR.TOO.BIG       ...TOO LOW
                        2640 *---LOAD FROM 1ST SECTOR---------
74F6- C8                2650         INY                   POINT AT FIRST PROGRAM BYTE
74F7- 20 A7 75          2660 .5      JSR LOAD.FROM.SECTOR
74FA- B0 19             2670         BCS ABORT             ...END OF LOAD
                        2680 *---LOAD REST OF FILE------------
74FC- A4 02             2690         LDY CAT.INDEX
74FE- C8                2700         INY
74FF- C8                2710         INY
7500- F0 13             2720         BEQ ABORT
7502- 84 02             2730         STY CAT.INDEX         NEXT TRACK/SECTOR
7504- B9 00 7E          2740         LDA TS.LIST,Y         TRACK
7507- F0 0C             2750         BEQ ABORT             ...END OF FILE
7509- BE 01 7E          2760         LDX TS.LIST+1.Y       SECTOR
750C- 20 68 75          2770         JSR RTS               READ IT
750F- 86 01             2780         STX PNTR+1            SET POINTER
7511- A0 00             2790         LDY #0
7513- F0 E2             2800         BEQ .5                ...ALWAYS
                        2810 *--------------------------------
7515- 4C 03 80          2820 ABORT   JMP $8003             WARMSTART ASSEMBLER
```

```
                        2830 *-------------------------------
                        2840 MENU.NEXT.SCREEN
7518- A5 07             2850         LDA DONE.FLAG
751A- F0 03             2860         BEQ .1
751C- 4C 1A 74          2870         JMP LOAD.MENU      START ALL OVER
751F- 20 96 75          2880 .1      JSR SETUP.SCREEN
7522- 4C 7B 74          2890         JMP GET.NEXT.FILE
                        2900 *-------------------------------
                        2910 ERR.EMPTY.FILE
7525- A0 56             2920         LDY #EM1
7527- 2C                2930         .HS 2C
                        2940 ERR.TOO.BIG
7528- A0 65             2950         LDY #EM2
752A- 20 30 75          2960         JSR PRINT.MSG
752D- 4C 03 80          2970         JMP $8003
                        2980 *-------------------------------
                        2990 PRINT.MSG
7530- B9 C4 75          3000 .1      LDA EMS,Y
7533- F0 06             3010         BEQ .2             00 IS END OF MESSAGE
7535- 20 ED FD          3020         JSR MON.COUT
7538- C8                3030         INY
7539- D0 F5             3040         BNE .1             ...ALWAYS
753B- 60                3050 .2      RTS
                        3060 *-------------------------------
                        3070 GETNUM
753C- 85 0A             3080         STA LIMIT
753E- 20 30 75          3090         JSR PRINT.MSG      PROMPT
7541- 20 0C FD          3100 .1      JSR MON.RDKEY
7544- C9 B1             3110         CMP #"1"
7546- 90 F9             3120         BCC .1             GO BACK IF TOO SMALL
7548- C5 0A             3130         CMP LIMIT
754A- B0 F5             3140         BCS .1             ...OR TOO LARGE
754C- 20 ED FD          3150         JSR MON.COUT       ECHO CHARACTER
754F- 49 B0             3160         EOR #"0"           EXTRACT VALUE
7551- 60                3170         RTS
                        3180 *-------------------------------
                        3190 READ.NEXT.CATALOG.SECTOR
7552- A9 0B             3200         LDA #$0B           RESTART INDEX
7554- 85 02             3210         STA CAT.INDEX
7556- 38                3220         SEC                IN CASE NO MORE SECTORS
7557- A0 02             3230         LDY #2
7559- B1 00             3240         LDA (PNTR),Y
755B- AA                3250         TAX                SECTOR
755C- 88                3260         DEY
755D- B1 00             3270         LDA (PNTR),Y       TRACK
755F- F0 06             3280         BEQ .1             END OF CATALOG
7561- 20 68 75          3290         JSR RTS            READ IT
7564- 86 01             3300         STX PNTR+1         PAGE IN BUFFER
7566- 18                3310         CLC                SIGNAL WE GOT A SECTOR
7567- 60                3320 .1      RTS
                        3330 *-------------------------------
                        3340 *    READ TRACK/SECTOR
                        3350 *      (A)=TRACK, (X)=SECTOR
                        3360 *      RETURNS (X)=PAGE OF BUFFER CONTAINING SECTOR
                        3370 *              CARRY SET IF ERROR
                        3380 *      CLOBBERS (A) AND (Y)
                        3390 *-------------------------------
                        3400 RTS
7568- A0 00             3410         LDY #0
756A- 0A                3420         ASL                TRACK*8
756B- 0A                3430         ASL
756C- 0A                3440         ASL
756D- 90 01             3450         BCC .1             BLOCK < $100
756F- C8                3460         INY                BLOCK > $0FF
7570- 0A                3470 .1      ASL                *2, MAKE ROOM FOR H/L FLAG BIT
7571- 1D 53 76          3480         ORA BLKTBL,X
7574- 6A                3490         ROR                H/L BIT TO CARRY
7575- 8D 67 76          3500         STA BLOCK
7578- 8C 68 76          3510         STY BLOCK+1
757B- A2 7C             3520         LDX /BLOCK.BUFFER
757D- 90 01             3530         BCC .2             LOWER HALF OF BLOCK
757F- E8                3540         INX                UPPER HALF OF BLOCK
7580- 20 00 BF          3550 .2      JSR $BF00
7583- 80 63 76          3560         .DA #$80.PARMLIST
7586- B0 01             3570         BCS .3             ...ERROR
7588- 60                3580         RTS
7589- 48                3590 .3      PHA                SAVE ERROR CODE
758A- A0 87             3600         LDY #EM5           "ERROR"
```

```
758C- 20 30 75 3610         JSR PRINT.MSG
758F- 68      3620          PLA
7590- 20 DA FD 3630         JSR MON.PRHEX      DISPLAY CODE
7593- 4C 03 80 3640         JMP $8003          SOFTLY BACK TO S-C MACRO
               3650 *-------------------------------
               3660 SETUP.SCREEN
7596- A9 14    3670         LDA #20            LINES PER SCREEN
7598- 85 04    3680         STA LINE.COUNT
759A- A9 C1    3690         LDA #"A"           START MENU WITH LETTER "A"
759C- 85 03    3700         STA MENU.LETTER
759E- 20 8E FD 3710         JSR MON.CROUT      THREE BLANK LINES
75A1- 20 8E FD 3720         JSR MON.CROUT
75A4- 4C 8E FD 3730         JMP MON.CROUT      RETURN THROUGH CROUT
               3740 *-------------------------------
               3750 *      RETURN .CS. IF END OF LOAD
               3760 *-------------------------------
               3770 LOAD.FROM.SECTOR
75A7- AD B6 75 3780         LDA LPTR+1         IS THERE ROOM FOR
75AA- C5 73    3790         CMP HIMEM          ANOTHER BYTE?
75AC- AD B7 75 3800         LDA LPTR+2
75AF- E5 74    3810         SBC HIMEM+1
75B1- B0 10    3820         BCS LFS2           NO, END OF LOAD
75B3- B1 00    3830         LDA (PNTR),Y
75B5- 8D 55 55 3840 LPTR    STA $5555
75B8- EE B6 75 3850         INC LPTR+1
75BB- D0 03    3860         BNE .1
75BD- EE B7 75 3870         INC LPTR+2
75C0- C8       3880 .1      INY
75C1- D0 E4    3890         BNE LOAD.FROM.SECTOR
75C3- 60       3900 LFS2    RTS
               3910 *-------------------------------
               3920 EMS
00-            3930 EM0     .EQ *-EMS
75C4- 8D       3940         .HS 8D
75C5- D4 D9 D0
75C8- C5 A0 CC
75CB- C5 D4 D4
75CE- C5 D2 A0
75D1- D4 CF A0
75D4- CC CF C1
75D7- C4 A0 C1
75DA- A0 C6 C9
75DD- CC C5 AC 3950         .AS -/TYPE LETTER TO LOAD A FILE,/
75E0- 8D       3960         .HS 8D
75E1- CF D2 A0
75E4- BC D3 D0
75E7- C1 C3 C5
75EA- BE A0 C6
75ED- CF D2 A0
75F0- CD CF D2
75F3- C5 A0 C6
75F6- C9 CC C5
75F9- D3 AC    3970         .AS -/OR <SPACE> FOR MORE FILES./
75FB- 8D       3980         .HS 8D
75FC- CF D2 A0
75FF- BC D2 C5
7602- D4 BE A0
7605- CF D2 A0
7608- BC C5 D3
760B- C3 BE A0
760E- D4 CF A0
7611- C1 C2 CF
7614- D2 D4 BA
7617- A0 A0    3990         .AS -/OR <RET> OR <ESC> TO ABORT:  /
7619- 00       4000         .HS 00
56-            4010 EM1     .EQ *-EMS
761A- 8D       4020         .HS 8D
761B- C6 C9 CC
761E- C5 A0 C9
7621- D3 A0 C5
7624- CD D0 D4
7627- D9       4030         .AS -/FILE IS EMPTY/
7628- 00       4040         .HS 00
65-            4050 EM2     .EQ *-EMS
7629- 8D       4060         .HS 8D
762A- C6 C9 CC
762D- C5 A0 C9
```

```
7630- D3 A0 D4
7633- CF CF A0
7636- C2 C9 C7 4070          .AS -/FILE IS TOO BIG/
7639- 00       4080          .HS 00
76-            4090 EM3      .EQ *-EMS
763A- A0 D3 CC
763D- CF D4 BA
7640- A0       4100          .AS -/ SLOT: /
7641- 00       4110          .HS 00
7E-            4120 EM4      .EQ *-EMS
7642- 8D       4130          .HS 8D
7643- C4 D2 C9
7646- D6 C5 BA
7649- A0       4140          .AS -/DRIVE: /
764A- 00       4150          .HS 00
87-            4160 EM5      .EQ *-EMS
764B- 8D       4170          .HS 8D
764C- C5 D2 D2
764F- CF D2 A0 4180          .AS -/ERROR /
7652- 00       4190          .HS 00
               4200 *-------------------------------
7653- 00 0E 0D
7656- 0C 0B 0A
7659- 09 08    4210 BLKTBL   .HS 00.0E.0D.0C.0B.0A.09.08
765B- 07 06 05
765E- 04 03 02
7661- 01 0F    4220          .HS 07.06.05.04.03.02.01.0F
               4230 *-------------------------------
               4240 PARMLIST
7663- 03       4250          .DA #3
7664- 60       4260 UNIT     .HS 60              DRIVE-1*8+SLOT*16
7665- 00 7C    4270          .DA BLOCK.BUFFER
7667- 02 00    4280 BLOCK    .DA 2
               4290 *-------------------------------
7669-          4300 TRACKS   .BS 21
767E-          4310 SECTORS  .BS 21
```

8086/8088 Cross Assembler

Use your Apple to learn 8086 programming! You can program for the IBM PC, the clones, and ALF's co-processor board without ever leaving the friendly environment of Apple DOS 3.3.

This easy-to-use cross assembler, based on the S-C Assembler II (Version 4.0), covers all the 8086 and 8088 instructions and all the addressing modes. Instruction mnemonics are based on the Microsoft 8086 assembler. Does not include newer S-C Assembler features like macros or the EDIT command.

Documentation covers the differences from standard S-C Assembler operation and syntax. Sample source programs help you become familiar with the assembler syntax.

With permission from S-C Software, XSM 8086/8088 is available to owners of any S-C Assembler for $80.00 post-paid. (No credit cards or purchase orders.)

Don Rindsberg
The Bit Stop
5958 S. Shenandoah Rd.
Mobile, AL 36608

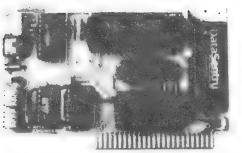
(205) 342-1653

APPLIED ENGINEERING

To a World of Compromise, We Make No Contribution

THE NEW TIMEMASTER II H.O.

- Absolutely, positively, totally PRO-DOS and DOS 3.3 compatible.
- Time in hours, minutes, seconds and milliseconds (the ONLY PRO-DOS compatible card with millisecond compatibility).
- 24 hour military format or 12 hour with AM/PM format.
- Date with year, month, day of week and leap year.
- Eight software controlled interrupts so you can run two programs at the same time (many examples are included).
- The only card recognized by both the DOS 3.3 and PRO-DOS versions of Apple Writer IIe.
- Appleworks will now time and date stamp all your data automatically.
- Compatible with ALL of Apple's languages. Many sample programs for machine code, Applesoft, CP/M and Pascal on 2 disks.
- On-board timer lets you time any interval up to 48 days long down to the nearest millisecond.
- Rechargeable nickle-cadmium battery will last over 20 years.
- Two BSR/serial ports for future expansion.



	PRO-DOS COMPATIBLE	INCLUDES DOS DATER	MILLISECOND TIME	YEAR DATA	LARGEST SAMPLE SOFTWARE	REMOTE SET PORT	BSR PORT	EMULATES ALL OTHER CLOCKS
H.O	YES	YES	YES	YES	YES	YES	YES	YES
BRAND A	NO	NO	YES	NO	NO	NO	NO	NO
BRAND T	NO	NO	NO	NO	NO	NO	NO	NO
BRAND M	NO	NO	NO	NO	NO	NO	NO	NO
BRAND P	YES	YES	NO	YES	NO	NO	NO	NO
BRAND S	NO	NO	NO	YES	NO	NO	NO	NO
BRAND C	YES	NO	NO	NO	NO	NO	YES	NO

Full emulation of all other clocks. Yes, we emulate Brand A, Brand T, Brand P, Brand C, Brand S and Brand M too. It's easy for the H.O. to emulate other clocks, we just drop off features. That's why the H.O. can emulate others, but none of the others emulate us. The Timemaster II H.O. will automatically emulate the correct clock card for the software you're using. You can also give the H.O. a simple command to tell it which clock to emulate. This is great for writing programs for those poor unfortunates who bought some other clock card.

REMOTE CONTROL

Our BSR X-10 interface option for the H.O. allows you to remotely control up to 16 lights and electrical appliances through your BSR X-10 home control system in your home or office. You're already wired because a BSR system sends its signals over regular 120 volt wiring. That means you can control any electrical device in your home or office without additional wiring.

PRICE $129.00 BSR Option (may be added later) $49.00

VIEWMASTER 80

There used to be about a dozen 80 column cards for the Apple. Now there is only ONE.

- TOTALLY Videx Compatible.
- 80 characters by 24 lines, with a sharp 7x9 dot matrix.
- On-board 40/80 soft video switch with manual 40 column override.
- Fully compatible with ALL Apple languages and software—there are NO exceptions.
- Low power consumption through the use of CMOS devices.
- All connections are made with standard video connectors.
- Both upper and lower characters are standard.
- All new design (using a new Microprocessor based C.R.T. controller) for a beautiful razor sharp display.
- The VIEWMASTER incorporates all the features of all other 80 column cards, plus many new improvements.

	PRICE	BUILT-IN SOFTSWITCH	SHIFT KEY SUPPORT	LOW POWER DESIGN	80 COLUMN HOME	7X9 DOT MATRIX	LIGHT PEN INPUTS	40 COLUMN OVERRIDE	INVERSE CHARACTERS
VIEWMASTER	$39	YES	YES	YES	YES	YES	YES	YES	YES
SUPERTERM	MORE	NO	YES	NO	NO	NO	NO	YES	YES
WIZARD 80	MORE	NO	NO	NO	NO	YES	NO	YES	YES
VISION 80	MORE	YES	YES	NO	NO	YES	NO	NO	NO
OMNIVISION	MORE	YES	YES	NO	NO	NO	NO	YES	NO
VIEWMAX 80	MORE	YES	YES	NO	NO	YES	NO	NO	YES
SMARTERM	MORE	YES	YES	NO	NO	YES	YES	YES	NO
VIDEX	MORE	NO	YES	YES	YES	YES	YES	NO	YES

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, Easywriter, Apple Writer II, VisiCalc, and all others. The VIEWMASTER 80 is THE MOST compatible 80 column card you can buy at ANY price!

PRICE $139.00

Z-80 PLUS

Now Includes New 4.0™ Software

Enter the CP/M world with the new Z-80 Plus card from Applied Engineering and introduce your Apple to thousands of new programs. Only the Z-80 Plus comes standard with the new 4.0 software, the most advanced system for running CP/M programs ever. Only CP/AM 4.0™ has advanced features like built-in disk emulation for popular memory expansion boards (those made by Apple and Applied Engineering and others) to give you a faster system with more storage. You also get menu driven utilities that are much easier to use than the older CP/M utilities so you can get down to all that great CP/M software faster. If you already own the Z-80 Plus, you can upgrade to the 4.0 software for only $29. The Z-80 Plus runs older CP/M programs too, down to Version 1.6 (2.2 is the most popular). With the Z-80 Plus you can run the largest body of software in existence. Simply plug the Z-80 Plus into any slot in your Apple. You'll have two computers in one and the advantages of both, all at an unbelievably low price.

- TOTALLY compatible with ALL CP/M software.
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with microsoft disks (no pre-boot required).
- Specifically designed for high speed operation in the Apple IIe (runs just as fast in the II+ and Franklin).
- Runs WORD STAR, dBASE II, TURBO PASCAL, FORTRAN-80, PEACHTREE and ALL other CP/M software with no pre-boot.
- A semi-custom I.C. and low parts count allows the Z-80 Plus to fly thru CP/M programs at a very low power level. (We use the Z-80A at a fast 4MHZ.)
- Does EVERYTHING the other Z-80 boards do, plus Z-80 interrupts.

PRICE $139.00

SUPER MUSIC SYNTHESIZER — END MOCKINGBOREDOM

- Complete 16 voice stereo music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away by inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.
- Now with new improved software for the easiest and the fastest music input system available anywhere.
- We gives you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Full control of attack, volume, decay, sustain and release.
- Our card will play notes from 30HZ to beyond human hearing.
- Automatic shutoff on power-up or if reset is pushed.
- Many many more features.
- Works in any slot of a IIe or II+ including slot 3 of a IIe.

PRICE $159.00

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in Apple IIe, II, II+ and Franklin. Applied Engineering also manufactures a full line of data acquisition and control products for the Apple: A/D converters and digital I/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle **THREE YEAR WARRANTY.**

APPLIED ENGINEERING

Call (214) 492-2027, 9 a.m. to 11 p.m. 7 days a week or send check or money order to P.O. Box 798, Carrollton, Texas 75006. MasterCard, Visa and C.O.D. welcome. No extra charge for credit cards. Texas residents add 5⅛% sales tax. Add $10.00 if outside U.S.A.

Review of M-c-T SpeedDemon.....Bob Sander-Cederlof

Is the Apple II a slow machine? Hey, it MUST be! After all, it is over 8 years old! It only has an 8-bit microprocessor! It only has a 1-MHz clock! It must be many times slower than today's PC clones, etc. Isn't it?

No.

The 6502 is inherently faster than most other microprocessors. An old rule of thumb had it that a 4-MHz Z-80 ran roughly the same speed as a 1-MHz 6502. Other factors, such as memory speeds, overhead for screen and keyboard, and disk I/O also influence the overall speed, often in favor of the venerable Apple

Some comparisons come to mind with machines from the past. Anyone remember MIT's "Whirlwind"? A long time ago, its speed was considered super. I'll bet it wasn't as fast as an Apple. According to the book, it had an upper limit of 2048 16-bit words of "high-speed" memory, and had a design limit of 50,000 instructions per second. In actual implementation, it only ever achieved 20,000 operations per second. And that was with a 1 MHz clock! The 6502 with a 1 MHz clock runs from 500,000 to 142,000 operations per second, depending on which ones you are doing. Probably an average of 250,000.

How about the Bendix G-15? It was the "personal" computer of the 1950's, roughly the size of a large refrigerator (much warmer though) and selling for only $50,000. Engineering firms bought them eagerly for their friendly features, amazing flexibility, capacity, and speed. Let's see.... G-15 had 2183 words of RAM, on a magnetic drum. 29 bits per word. Most operations were measured in milliseconds. A floating point interpretive package, called Intercom 500 (or 1000 for double precision), could almost keep up with the typewriter (an IBM Executive, the primary user I/O device). Paper tape cassettes served as handy off-line storage devices.

Some other popular systems were considered fast with memory cycle times over ten microseconds per byte. Fast enough to support several users in a timesharing environment, compile large Fortran programs, and manage large businesses. And usually with smaller than 128K bytes of RAM. Or "core", as we called it in those days.

Nevertheless, Apples often seem slow. Because we ask them to do a lot, and don't want to wait around while it is done. And tolerable waiting times one day seem intolerable the next, because we get used to it. Remember when a trip around the world in 80 days seemed impossibly fast?

Perceived necessity being a prime motivator for innovation, several methods for dramatically accelerating Apples have been developed. Titan Technologies markets the Accelerator, and Microcomputer Technologies (McT) the SpeedDemon. These both promise "up to" 3 5 times faster running speed, and actually deliver an average of over 2 times faster.

NEW DON LANCASTER RELEASES

Available ONLY from Synergetics. All software open and unlocked.

ABSOLUTE RESET (IIc/old IIe/new IIe) $19.50

Get back in total control. No more hole blasting! Access any part of any program at any time for any reason. Passes all diagnostics. Invisible till activated. Includes EPROM burner adapter details, service listings, and a free bonus book.

APPLEWRITER™ TOOLKITS (DOS 3.3 or ProDOS) $39.50

EIGHT diskette sides include a complete disassembly script; source code capturing; self-prompting glossaries; Diablo microjustify and proportional space, patches including NULL, shortline, Grappler, IIc detrashing, many others; answers to most help line questions; two columns; WPL secrets; space on disk; keyword indexing; multi and even-odd headers; bunches more.

Both TOOLKITS (sixteen disk sides) $59.50

APPLEWORKS™ DISASSEMBLY SCRIPT $49.50

TEN diskette sides chock full of Applework's innermost secrets, using Don's unique and powerful "tearing method". Primarily for gonzo hackers and definitely NOT for the faint of heart.

LASERWRITER/APPLEWRITER UTILITIES $39.50

Two volume package gives you unmatchably superb IIe text and graphics. Included are automatic formatters, boxes and fancy borders, daisywheel changers, envelope and label routines, unbelievably fast formletters, grids and rulers, HIRES converters, DC1 patches, demos, justify routines, self-prompting glossaries, help screens, a sign shop, outlined text, more.

AUTOGRAPHED LANCASTER BOOKS:

Apple IIe Assembly Cookbook	$21.50
All About Applewriter IIe	$14.50
Enhancing Your Apple II & IIe, (Volume I or II)	$15.50
Micro Cookbook (Volume I or II)	$15.50
CMOS Cookbook	$14.50
TTL Cookbook	$12.50
Incredible Secret Money Machine	$7.50
Complete book and software list	(FREE)

COMPANION DISKETTES:

For Enhancing Your Apple II, Volume I	$19.50
For Enhancing Your Apple II, Volume II	$19.50
For Don Lancaster's Assembly Cookbook	$19.50

SYNERGETICS
746 First Street
Box 809-SC
Thatcher, AZ, 85552

FREE
VOICE HELPLINE
(602) 428-4073

Appleworks, Applewriter, and ProDOS are registered trademarks of Apple Computer.
VISA and MASTERCHARGE accepted. Please - no COD, foreign, or purchase orders.

We have wanted to try one of these boards for years. The price was too high and our faith too low, so we never bought one. Recently the price has dropped considerably, and reports from friends using them have increased our faith. When McT offered to loan us one for a month, we had no more resistance at all.

Imagine this scenario: the card arrives by UPS at noon. Thirty seconds later we have it in our hands, and are trying to find an Apple with at least one empty slot. Despairing of that, we take out a card and make room for the SpeedDemon in our //e. We turn on the //e, load up the S-C Macro Assembler, and proceed to assemble the biggest program we have. Wow! That's fast!

We promptly ran a lot of speed tests, timing various programs we commonly use around here:

S-C Word Processor			
Load 89 sectors	6 8	5.5	1.2
Search /###/	10.4	3 3	3.2
Replace /85/##/	8.3	2.8	3 0
Mail Label System (primarily Applesoft)			
Load 48 sectors	23.7	13.8	1.7
Sort - last name	140.6	49.1	2.9
Sort - zip code	56 0	20.0	2.8
S-C Macro Assembler			
Assemble 771 lines	7.2	3.0	2.4
AppleWorks Data Base			
Load 47K	25.7	25.0	1.0+
Sort - last name	2.2	1.0	2.2
Sort - zip code	5.0	2.0	2.5
AppleWorks Spreadsheet			
Load 35K	20.3	19.3	1.1
Recalculate	14.9	6 6	2.3
Insert 9 rows	4.9	1 8	2.7

In a review by Lee The, Personal Computing, Jan 85, the Apple with SpeedDemon was compared to a Compaq PC. Lee compared the systems using word processors on the two machines. The accelerated Apple ran faster in most cases, except when disk I/O was involved. In one case, even an un-accelerated Apple ran faster; the SpeedDemon to Compaq ratio was 4.4!

To summarize, the SpeedDemon really does make your software run faster The absolute maximum speedup factor is 3 5, but no "real" program would achieve it. The two things that keep you from reaching 3 5 are I/O and memory.

Some I/O cards, notably the disk interface, use software timing. If you speed up the processor while trying to read or write the disk, you are in trouble. SpeedDemon automatically slows down to normal Apple speed when you access slot 6. Jumpers on the card allow you to do the same for slots 4 and 5.

I have a disk controller in slot 7 in one of my Apples; I cannot read or write to disks using that controller when the SpeedDemon is active.

Old Apple serial interface cards used software timing loops to convert a byte to a bit stream at a given baud rate. These cards normally were placed in slots 1 or 2, and thus would not be compatible with the SpeedDemon. Modem cards sometimes use software timing for dialing, and they would not work right if accelerated. Any sound effects created through the Apple speaker will be raised way up in pitch. Music cards which depend on timing loops will make a whole new kind of sound.

The card can be turned off in two ways, so the above problem areas can be circumvented. During the power up cycle you have about two seconds during which you may tap the ESCAPE key. If you do, the card will be turned off. Then you hit ctrl-RESET to go into a normal boot. Another way to turn off the card is to store anything into $C05B (POKE 49243,0). After the POKE the Apple will lock up; when you hit ctrl-RESET it will come back in normal speed. There is no way to turn the card back on without turning off the Apple. (Some of you can probably find a way to re-wire it so it could be turned back on.)

The other way the card slows down is during memory access. Apple memory can only be accessed at a 1 MHz rate, so the processor can spend time waiting for memory. SpeedDemon has a 4096-byte cache memory which can run at a full 3.58 MHz rate. The cache is implemented with 4 static RAM chips, providing 8192 bytes of RAM. These are paired so that you get 4096 data bytes and 4096 address bytes. Whenever you read a byte from RAM or ROM, the low-order 12 bits of the address select one of thes 4096 byte pairs. The high 4 bits of the address are compared to the 4 bits in the cache; if they are the same then the data in the cache is presumed to be the data you want. If not, the processor will wait for Apple's memory to read, and then update the cache with the result. Something like that, anyway. Stores into memory always slow down to a 1 MHz rate, because the stores MUST be performed in real RAM, not just cache RAM.

I might have been talking through my hat in the above paragraph. There is no technical documentation available on the SpeedDemon, so I am just deducing the way it works from external appearances.

The Titan Accelerator card has a full 64K RAM, rather than a cache. It is therefore a little bit faster. Reports from those who have tried both indicate Titan is only about 10 percent faster, if that much. Of course you could design artificial situations in which the difference would be much more dramatic. Personally I think I would rather have the cache. And also the cash, since SpeedDemon costs about $25 less.

Titan's card draws about 300 ma at 5 volts, SpeedDemon draws about 600 ma. Titan's card uses more CMOS, and is more sensitive to static electricity.

SpeedDemon uses a 65C02, so you have the additional opcodes and address modes of this enhanced 6502 chip available. I believe you could romove the 65C02 plug a 65802 into the socket and gain even greater enhancements. You would have to have a 65802 rated at 4MHz, but the ones I have are only 2 MHz chips.

There are five PLA's on the SpeedDemon. At least some of these are used to keep track of whatever bank switching you do with Apple's RAM and ROM. Somehow they are able to keep track of the RAMWORKS card too, so the cache doesn't get confused even with a megabyte of RAM. I worry about using it with my STB128 card, or the other cards of the type. Boards which store into Apple RAM using DMA transfer will possible give trouble. I don't know for certain because I don't have any.

I also worried about compatibility with QuikLoader. Both QL and SD want to take control of the bus on power up or reset. Both substitute their own firmware for whatever is plugged into the mother board. Sure enough, when I tried them both in the same machine they did not work. On power up both cpu's began to operate. SD drew its hi-res graphic logo, and then died. QL died too. Take either card out, and all is well.

Speaking of firmware, I should mention that there is a 2716 with 2K of firmware on the SpeedDemon. When you power up or hit ctrl-RESET the firmware on the card takes control. It sets a bunch of //e soft switches, in case it is in a //e, and then looks at the power-up bytes to see whther this is a RESET or power up. (Remember the power up bytes at $3F3 and $3F4? These bytes will be random when you first turn on your Apple, but during initialization they are set so that the exclusive-or of the two bytes is $A5.) If SpeedDemon thinks you have pressed ctrl-RESET, it copies a short (21-byte) program from its own ROM down to $1D0 and jumps to it. The program turns off the SpeedDemon ROM (by storing at $C800) and then uses a loop to make sure the cache doesn't contain misleading information (I call this action TRASHING the CACHE). Then it jumps to Apple's normal reset code.

If SpeedDemon thinks it is power-up time, because the "eor" the bytes at $3F3 and $3F4 is not $A5, it trashes the cache and copies a large program down to RAM at $1000 through $17FF. Then it trashes the cache again, clears the text screen, and jumps to $1000. The copied code at $1000 turns off the firmware ROM, clears the hi-res screen, switches on hi-res graphics, and draws the SpeedDemon logo. This all takes about two seconds. Then it reads the keyboard to see whether you have typed an ESCAPE, a "1", or a "T". ESCAPE signals SpeedDemon you want to run at normal Apple speed, so it shuts itself off. The other codes cause self-testing code to be executed.

I had a lot of fun figuring out the firmware. It so happens they purposely arranged all the bits in the EPROM in reverse order, so that I had to write a program to flip the bytes around before disassembling the code. I guess it was an attempt to frustrate reverse engineering. I think they should

12 Good Reasons Why RAMWORKS™ Is The Best Expansion Card For Your IIe

1 APPLEWORKS MEMORY Even though Ramworks enhances and expands a VAST ARRAY of other programs, Appleworks is our claim to fame. A 64K Ramworks will ADD 46K to your available desktop memory, a 128K Ramworks will ADD 91K, a 256K Ramworks will ADD 182K, and a 512K Ramworks will ADD 364K and a 1 meg Ramworks will give you nearly an 800K desktop. And it's all done automatically! When you plug in more memory chips into your Ramworks card, Appleworks will find them automatically. Ramworks also increases the maximum number of records from 1350 to 4300.

2 APPLEWORKS SPEED AND POWER Ramworks does more than just increase the desktop memory (as if that weren't enough). With Ramworks, Appleworks will be able to run up to 20 times faster. If you buy a 256K or larger Ramworks card, Appleworks will automatically load itself in Ramworks. This greatly increases the speed at which Appleworks operates by eliminating all that nasty, time consuming disk access on Drive 1. These are but a few reasons why we say that Ramworks is Appleworks best friend.

3 EXPANDABILITY Ramworks was designed with the future in mind, as your needs increase, so can Ramworks. Clear instructions show you how to plug in more memory (up to 1 meg).

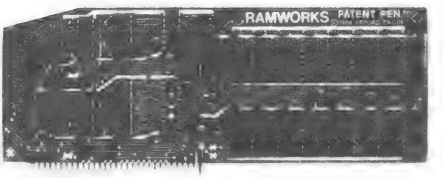
4 SPEED Today, as programs become more and more sophisticated, they inevitably become larger. And many of today's best selling programs (like Appleworks) won't fit in a 128K Apple, so many of these new larger programs continually go back to disk in search of more data. With Ramworks, you can have enough memory so that the entire program will be loaded into Ramworks' memory. This greatly increases the speed of software because your disk runs at 300 RPM, but Ramworks operates at the speed of light!

5 COLOR The same slot that's used for memory expansion is also the slot that's used for RGB color display, so all those lesser memory cards of yesterday make you decide in advance if you want RGB color. Only Ramworks lets you decide later to add RGB color. For only $129, an RGB option can be added to Ramworks to give you double high resolution color graphics and 80 column text. All with a razor sharp, vivid brilliance that's unsurpassed in the industry. The RGB option does not waste another valuable slot, but rather plugs into the back of Ramworks and attaches to any Apple compatible monitor. Remember, you can order the RGB option with your Ramworks or add it on at a later date.

6 COMPATIBILITY, OF THE SOFTWARE KIND Programs like Appleworks, Magic Office System, Flashcalc, The Spread Sheet, Diverse-A-Dos, Supercalc, Magicalc and many others automatically recognize all or most of Ramworks memory (512K is average). The simple fact is that Ramworks is compatible with more off-the-shelf software than any other RAM card. Ramworks is 100% compatible with ALL software written for the Apple 80 column and extended 80 column card. Additionally, Ramworks can emulate other RAM cards so software written for other cards will run without modification. Software written for RAMWORKS will not work on other cards. We can emulate others, but others can't emulate us.

7 COMPATIBILITY, OF THE HARDWARE KIND Unlike others, Ramworks is fully compatible with hardware add on's from other companies, like the Sider and Profile hard disks. And Ramworks was designed in accordance with the official expansion rules defined by Apple so you don't have to worry about compatibility problems. As you continue to expand and make your Apple more powerful with other expansion products from Applied Engineering, you'll appreciate how each product has extra features designed to work with Ramworks and other products to give you a total performance package that is more powerful than the sum of its parts.

8 IT SELLS THE MOST Popularity translates into great software support because software companies can't support all RAM cards, they can only support the ones their customers are likely to own. And software companies appreciate the fact that when they write software for Ramworks in the IIe.



they're also writing software for our memory expansion card for the IIc, Z-RAM. And our customer list reads like the Who's Who of Apple computing with just about every software company in the land buying one, including Apple Computer (in the hundreds), Rupert Lissner, and Steve Wozniak (we didn't give one to Mr. Wozniak just to use his name, 2 one meg Ramworks were paid for at full price).

9 IT'S FROM APPLIED ENGINEERING Unlike most of the competition, we only make accessories for Apple, so we'll never spend your money on IBM product research. Applied Engineering's years of experience and wide product line really pays off, and because of our high sales levels we buy most of our I.C. chips factory direct. So don't let our low prices fool you, they're caused by high volume production. That's why we can offer the most memory for the least money. Guaranteed!

10 IT'S GOT IT ALL

- ☑ Sharp 80 Column Text
- ☑ Double high resolution graphics (with or without RGB option)
- ☑ User Expandable to 1 Megabyte
- ☑ Can Use 64K or 256K RAMS in any combination
- ☑ Adds Memory to Appleworks
- ☑ Accelerates Appleworks
- ☑ 100% Compatibility with All IIe software
- ☑ RAM Disk software available, compatible with Applesoft, PRO-DOS, DOS 3.3, and PASCAL ($29)
- ☑ RAM Disk available for CP/M ($29). (This program is included with our CP/M card)
- ☑ Visicalc preboot available ($29)
- ☑ RGB option
- ☑ Takes only one slot
- ☑ 3 year no hassle warranty

11 THE PATENT OFFICE HAS ONE There are many advanced features on Ramworks, but two parts of the design are so advanced we applied for patents. One patent application deals with our ultra fast, ultra smooth 80 column screen display, and the other patent application deals with our ingenious way of dramatically reducing the power and heat of memory chips and improving reliability at the same time.

12 HERE TODAY, HERE TOMORROW In the seven years we've been making products for the Apple, we've seen a lot of companies come and go. Although nothing is forever, we're growing, expanding and we're profitable. And we are totally committed to Apple computing, which means you'll never run out of things to do with Ramworks. Or for that matter, reasons to buy one.

Ramworks™ with 64K	**$179**
Ramworks™ with 128K	**$249**
Ramworks™ with 256K	**$299**
Ramworks™ with 512K	**$399**
Ramworks™ with 1 MEG	**$649**
RGB Option (can be added later)	**$129**

Call (214) 241-6060
9 a.m. to 11 p.m. 7 days a week or send check or money order to:
Applied Engineering, P. O. Box 798, Carrollton, Texas 75006

MasterCard VISA Visa and
C.O.D. welcome. No extra charge for credit cards. Texas Residents add 5¼% sales tax. Add $10.00 if outside U.S.A.

APPLIED ENGINEERING
"We Set the Standard"

have re-arranged the address lines too, if they really are worried about it.

If all the above makes you want to rush right out and buy one, the price is $295 from Microcomputer Technologies (McT), at 1745 21st St., Santa Monica, CA 90404. Their phone number is (213) 829-3641 If you are a member of Call APPLE, they are selling the SpeedDemon card for only $199. The name on the card has been changed to "Mach 3.5", but it is the same as SpeedDemon. Call them at (206) 251-5222. Since the Call APPLE price is as close to wholesale price as we can get, we will not be trying to sell this board at S-C Software.

By the way, Call APPLE's ad contains a warning: "Mach 3.5 is not compatible in speedup mode with Saturn, Legend, Prometheus expansion memory cards with programs that make use of the extra banks on these cards. A compatible version of Mach 3.5 may be specially ordered."

Multi-Level ProDOS Catalog.....Bob Sander-Cederlof

Last week I looked through some old piles of papers and came across a program by Greg Seitz, dated Dec 20, 1983 It was attached to a set of ProDOS Tech Notes, and Greg apparently worked at Apple at that time.

Greg's program lists the filenames of an entire ProDOS directory, showing the whole tree. It shows directory files by printing a slash in front of the filename, and shows the level by indenting. For example, a typical listing might look like this:

```
PRODOS
BASIC.SYSTEM
/UTILITIES
 HELPER
 DOER
 /MORE
  WHATEVER
  AND.ANOTHER
 TEXT.FILE
ANOTHER
```

A listing like this can be a big help in finding things on a large hard disk. The program can also be extended in many ways. One that comes to mind immediately is to print the rest of the CATALOG information as well as the file names. Another is to create a complete CATALOG MANAGER utility, which would permit re-arranging the filenames, promoting and demoting files, and so on.

I typed in Greg's program, and then I rewrote it. The listing that follows bears very little resemblance to his code, but I do thank him for the help in getting started.

The program assumes a prefix has been set. If there is no prefix, you will get a beep and no listing. If there is a prefix, and the directory named is online, the listing will begin with that directory. Another enhancement would be to display the current prefix, and allow accepting it or changing it before starting the filename listing.

If we were always starting with the volume directory, it would be a little easier The volume directory always starts in block 2. However, since we are able to start with any directory, we do not know where it starts. ProDOS allows you to read a directory, and we can get the first block of any directory by using MLI to open the directory file.

Lines 1100-1120 read the current prefix into a buffer. The lines 1130-1150 open that file. Although I have never seen it in the books, apparently OPEN also reads the first block. After the OPEN call, BUFFER ONE contains the first block of the directory file. Unless we are willing to do a complete search without ProDOS's help, this is the only way I know of to find the first block of a directory file (other than the volume directory).

Since the only reason to OPEN the directory file was to read the first block, lines 1180-1200 close it again. If any of these MLI calls don't go through, line 1210 will ring the alarm and stop.

Lines 1230-1260 start up the directory listing. The first block ONLY will be in BUFFER.ONE. All subsequent blocks will be read into BUFFER.TWO. In order to make the LIST.DIRECTORY program completely recursive, it is called with the buffer address in a zero-page pointer. SETUP.NEXT.BLOCK also gets the next block pointer from the buffer and saves it in NEXT.BLOCK.

LIST.DIRECTORY is really quite simple, in spite of its size. Its main function is to print a list of filenames. Each filename is preceded by a number of blanks, determined by NEST.LEVEL. NEST.LEVEL is incremented at line 1290, each time LIST.DIRECTORY is called. If a file listed happens to be a directory file, LIST.IDRECTORY saves all the pointers and counters on the stack and then calls itself. When the subdirectory's files have all been listed, that recursive call of LIST.DIRECTORY will return, the pointers and counters can be unstacked, and the listing can continue.

The format of the information in a directory is detailed quite well in both "Beneath Apple ProDOS" and "Apple ProDOS Advanced Features". (We recommend and sell both books.) The first four bytes of each block are two block numbers: that of the previous block, and that of the next block, in the same directory. This allows scanning in both forward and reverse directions through a directory We will only use the next-block pointers in our program. After the block numbers there are 13 descriptors of 39 bytes each. The first descriptor in a directory describes the directory itself, and the rest describe files.

For some reason Apple was not quite sure that it would always use 13 39-byte descriptors, so they stored these two numbers in the directory descriptor. Anyone who access a directory is supposed to look up these two numbers and use them, just in case Apple decides to change them someday. The directory descriptor also contains an active file count. When a file is deleted this count is decremented, but the file descriptor remains. We use the active file count to determine when we reach the end of a directory. Lines 1300-1360 pick up the bytes per descriptor, descriptors per block, and active file count and save them

Lines 1370-1450 set up PNTR to point at the first file descriptor, which follows the directory header. CURRENT.ENTRY.NUMBER will count up to 13, so we will know when it is time to read another block. We start at 2, because the first block uses the first descriptor for the header. We also clear the file count.

Lines 1460-1500 check for the special case of an empty directory. If there are no active files, we are finished.

Lines 1510-1750 print out the file name from the current file descriptor. The first byte of a descriptor contains a code for the type of file in the first nybble, and the length of the file name in the second nybble. If both are zero, the file has been deleted. The other legal values are $1x, $2x, and $3x to signify a seedling, sapling, or tree file, respectively; and $Dx to signify a directory file. All we care about is whether is a directory file or not, and how long the file name is.

If it is a directory file, lines 1760-2100 will be executed. Lines 1760-1860 push the counters and pointers on the stack. Lines 1870-1930 read in the first block of the sub-directory. Line 1950 calls LIST.DIRECTORY to list the subdirectory. After it is finished, line 1960 will decrement the nesting level. Lines 1970-2060 unstack the pointers and counters. If we were still in the first block of the highest level directory (where we started), we do not need to read the block again: it is still in BUFFER.ONE. Otherwise, lines 2070-2100 read the block back in. If we did not care how much memory we used, we could make this program a lot faster by using more buffers. We could have a different buffer for each level, so that blocks would never have to be re-read.

Lines 2110-2210 count the file just listed, and then check to see if our count is the same as the active file count from the directory header. If so, we are finished.

If we are not finished, lines 2220-2290 bump the pointer into the directory block by the size of a descriptor entry. If we are still in the same block, that is all that we need to do. If not, lines 2350-2420 read in the next block and set things up for it. Then it's back to the top again for the next file name!

We hope some time in the not-so-distant future to be able to write a complete catalog manager program like I started to describe back at the beginning of this article. Some of you are using Bill Morgan's CATALOG ARRANGER for DOS 3.3, and this would be an equivalent utility for ProDOS. We're not quite ready yet, but this program is a step in the right direction.

```
                      1000 *SAVE S.RECURCAT
                      1010 *-------------------------------
BF00-                 1020 MLI     .EQ $BF00
BF30-                 1030 DEVNUM  .EQ $BF30
FBDD-                 1040 BELL    .EQ $FBDD
FD8E-                 1050 CROUT   .EQ $FD8E
FDED-                 1060 COUT    .EQ $FDED
EB-                   1070 PNTR    .EQ $EB AND EC
                      1080 *-------------------------------
                      1090 CAT
0800- 20 00 BF        1100         JSR MLI           GET CURRENT PREFIX
0803- C7 2A 09        1110         .DA #$C7.P.PREFIX
0806- B0 16           1120         BCS .1            ...ERROR
0808- 20 00 BF        1130         JSR MLI           OPEN THE DIRECTORY
080B- C8 2D 09        1140         .DA #$C8,P.OPEN     AND READ FIRST BLOCK
080E- B0 0E           1150         BCS .1            ...ERROR
0810- AD 30 BF        1160         LDA DEVNUM        SET UP READ MLI BLOCK
0813- 8D 36 09        1170         STA R.DEVNUM
0816- 20 00 BF        1180         JSR MLI           CLOSE THE DIRECTORY
0819- CC 33 09        1190         .DA #$CC,P.CLOSE
081C- 90 04           1200         BCC .2            ...NO ERROR
081E- 20 DD FB        1210 .1      JSR BELL          INDICATE ERROR
0821- 60              1220         RTS
```

```
0822- A9 00    1230 .2     LDA #0        BUFFERS ON PAGE BOUNDARIES
0824- 8D 44 09 1240        STA NEST.LEVEL    START AT TOP LEVEL
0827- A0 0A    1250        LDY /BUFFER.ONE   POINT TO NEXT BLOCK
0829- 20 18 09 1260        JSR SETUP.NEXT.BLOCK
               1270 *-------------------------------
               1280 LIST.DIRECTORY
082C- EE 44 09 1290        INC NEST.LEVEL    DROP TO NEXT LEVEL
               1300 *---GET DIR DATA----------------
082F- A0 26    1310        LDY #38
0831- B1 EB    1320 .1     LDA (PNTR),Y          GET: BYTES.PER.ENTRY....35
0833- 99 18 09 1330        STA BYTES.PER.ENTRY-35,Y  ENTRIES.PER.BLOCK..36
0836- 88       1340        DEY                       FILE.COUNT......37,38
0837- C0 23    1350        CPY #35
0839- B0 F6    1360        BCS .1
               1370 *---POINT TO FIRST FILE NAME-----
083B- A9 02    1380        LDA #2            SKIP OVER DIR HEADER
083D- 8D 41 09 1390        STA CURRENT.ENTRY.NUMBER
0840- 0A       1400        ASL               A=4. CLEAR CARRY
0841- 6D 3B 09 1410        ADC BYTES.PER.ENTRY
0844- 85 EB    1420        STA PNTR          POINT AT FIRST NAME
0846- A9 00    1430        LDA #0            START FILE COUNT
0848- 8D 3F 09 1440        STA CURRENT.FILE.COUNT
084B- 8D 40 09 1450        STA CURRENT.FILE.COUNT+1
               1460 *---STOP IF NO ACTIVE FILES------
084E- AD 3D 09 1470        LDA ACTIVE.FILE.COUNT
0851- 0D 3E 09 1480        ORA ACTIVE.FILE.COUNT+1
0854- D0 01    1490        BNE .2          ...AT LEAST ONE FILE
0856- 60       1500        RTS             ...END OF DIRECTORY
               1510 *---PRINT FILE NAME--------------
0857- A0 00    1520 .2     LDY #0            POINT TO TYPE/LENGTH
0859- B1 EB    1530        LDA (PNTR),Y
085B- F0 6C    1540        BEQ .8            0 = DELETED FILE
085D- 29 0F    1550        AND #$0F          ISOLATE NAME LENGTH
085F- AA       1560        TAX               X = #CHARS IN NAME
0860- AC 44 09 1570        LDY NEST.LEVEL    NUMBER OF LEADING BLANKS
0863- A9 A0    1580        LDA #" "
0865- 20 ED FD 1590 .3     JSR COUT          INDENT BY DIRECTORY LEVEL
0868- 88       1600        DEY
0869- D0 FA    1610        BNE .3
086B- B1 EB    1620        LDA (PNTR),Y      GET TYPE/LENGTH
086D- 48       1630        PHA               1L, 2L, 3L, OR DL
086E- 10 05    1640        BPL .4            ...NOT DIR FILE
0870- A9 AF    1650        LDA #"/"          DIR FILE, PRINT A SLASH
0872- 20 ED FD 1660        JSR COUT
0875- C8       1670 .4     INY               PRINT THE FILE'S NAME
0876- B1 EB    1680        LDA (PNTR),Y
0878- 09 80    1690        ORA #$80
087A- 20 ED FD 1700        JSR COUT
087D- CA       1710        DEX
087E- D0 F5    1720        BNE .4
0880- 20 8E FD 1730        JSR CROUT
0883- 68       1740        PLA               GET TYPE/LENGTH AGAIN
0884- 10 3B    1750        BPL .7            ...NOT DIR FILE
               1760 *---PUSH DATA ON STACK-----------
0886- A5 EC    1770        LDA PNTR+1        SAVE POINTER IN CURRENT BLOCK
0888- 48       1780        PHA
0889- A5 EB    1790        LDA PNTR
088B- 48       1800        PHA           SAVE:  R.BLOCK
088C- A2 00    1810        LDX #0               BYTES.PER.ENTRY
088E- BD 39 09 1820 .5     LDA PUSH.VARS.X      ENTRIES.PER.BLOCK
0891- 48       1830        PHA                  ACTIVE.FILE.COUNT
0892- E8       1840        INX                  CURRENT.FILE.COUNT
0893- E0 0B    1850        CPX #PUSH COUNT      CURRENT.ENTRY.NUMBER
0895- D0 F7    1860        BNE .5               NEXT.BLOCK
               1870 *---READ HEADER OF SUBDIR--------
0897- A0 12    1880        LDY #$12          POINT AT KEYBLOCK POINTER
0899- B1 EB    1890        LDA (PNTR),Y      GET HIGH BYTE
089B- AA       1900        TAX
089C- 88       1910        DEY
089D- B1 EB    1920        LDA (PNTR),Y      GET LOW BYTE
089F- 20 08 09 1930        JSR READ.NEXT.BLOCK
               1940 *---RECURSIVE CALL---------------
08A2- 20 2C 08 1950        JSR LIST.DIRECTORY
08A5- CE 44 09 1960        DEC NEST.LEVEL    POP TO HIGHER LEVEL
               1970 *---POP DATA OFF STACK-----------
08A8- A2 0B    1980        LDX #PUSH.COUNT   GET BLOCK OF VARS
08AA- 68       1990 .6     PLA
08AB- 9D 38 09 2000        STA PUSH.VARS-1,X
08AE- CA       2010        DEX
```

```
08AF- D0 F9     2020          BNE .6
08B1- 68        2030          PLA
08B2- 85 EB     2040          STA PNTR            GET KEYBLOCK POINTER
08B4- 68        2050          PLA
08B5- 85 EC     2060          STA PNTR+1
08B7- C9 0C     2070          CMP /BUFFER.TWO     IS BLOCK IN BUFFER.TWO?
08B9- 90 06     2080          BCC .7              ...NO, DON'T NEED TO READ
08BB- 20 00 BF  2090          JSR MLI             ...YES. MUST READ THE BLOCK
08BE- 80 35 09  2100          .DA #$80.P.READ
                2110 *---COUNT THE FILE--------------
08C1- EE 3F 09  2120 .7       INC CURRENT.FILE.COUNT
08C4- D0 03     2130          BNE .8
08C6- EE 40 09  2140          INC CURRENT.FILE.COUNT+1
                2150 *---SEE IF THAT WAS LAST FILE----
08C9- AD 3F 09  2160 .8       LDA CURRENT.FILE.COUNT
08CC- CD 3D 09  2170          CMP ACTIVE.FILE.COUNT
08CF- AD 40 09  2180          LDA CURRENT.FILE.COUNT+1
08D2- ED 3E 09  2190          SBC ACTIVE.FILE.COUNT+1
08D5- 90 01     2200          BCC .9              ...NOT LAST FILE
08D7- 60        2210          RTS                 ...END OF DIRECTORY
                2220 *---ADVANCE PNTR TO NEXT ENTRY---
08D8- 18        2230 .9       CLC
08D9- A5 EB     2240          LDA PNTR            GET RESULT IN Y,X
08DB- 6D 3B 09  2250          ADC BYTES.PER.ENTRY
08DE- AA        2260          TAX
08DF- A5 EC     2270          LDA PNTR+1
08E1- 69 00     2280          ADC #0
08E3- A8        2290          TAY
                2300 *---ARE WE STILL INSIDE BLOCK?---
08E4- AD 41 09  2310          LDA CURRENT.ENTRY.NUMBER
08E7- EE 41 09  2320          INC CURRENT.ENTRY.NUMBER
08EA- CD 3C 09  2330          CMP ENTRIES.PER.BLOCK
08ED- 90 12     2340          BCC .10             ...INSIDE SAME BLOCK
                2350 *---READ NEXT BLOCK-------------
08EF- AD 42 09  2360          LDA NEXT.BLOCK
08F2- AE 43 09  2370          LDX NEXT.BLOCK+1
08F5- 20 08 09  2380          JSR READ.NEXT.BLOCK
08F8- A9 01     2390          LDA #1              START WITH FIRST ENTRY
08FA- 8D 41 09  2400          STA CURRENT.ENTRY.NUMBER    IN NEW BLOCK
08FD- A2 04     2410          LDX #4              SKIP OVER BLOCK NUMBERS
08FF- A0 0C     2420          LDY /BUFFER.TWO
0901- 86 EB     2430 .10      STX PNTR            NEW PNTR VALUE
0903- 84 EC     2440          STY PNTR+1
0905- 4C 57 08  2450          JMP .2              ...TO LIST NEXT FILENAME
                2460 *--------------------------------
                2470 READ.NEXT.BLOCK
0908- 8D 39 09  2480          STA R.BLOCK         BLOCK # IN X,A
090B- 8E 3A 09  2490          STX R.BLOCK+1
090E- 20 00 BF  2500          JSR MLI             READ THE BLOCK
0911- 80 35 09  2510          .DA #$80.P.READ
0914- A9 00     2520          LDA #BUFFER.TWO     WE USED BUFFER.TWO
0916- A0 0C     2530          LDY /BUFFER.TWO
                2540 SETUP.NEXT.BLOCK
0918- 85 EB     2550          STA PNTR            PNTR FROM Y,A
091A- 84 EC     2560          STY PNTR+1
091C- A0 02     2570          LDY #2              GET NEXT BLOCK #
091E- B1 EB     2580          LDA (PNTR),Y
0920- 8D 42 09  2590          STA NEXT.BLOCK
0923- C8        2600          INY
0924- B1 EB     2610          LDA (PNTR),Y
0926- 8D 43 09  2620          STA NEXT.BLOCK+1
0929- 60        2630          RTS                 RETURN
                2640 *--------------------------------
092A- 01        2650 P.PREFIX    .DA #1
092B- 00 0C     2660             .DA BUFFER.TWO
                2670 *--------------------------------
092D- 03        2680 P.OPEN      .DA #3
092E- 00 0C     2690             .DA BUFFER.TWO
0930- 00 0A     2700 OPENBUF     .DA BUFFER.ONE
0932- 00        2710             .DA #0
                2720 *--------------------------------
0933- 01        2730 P.CLOSE     .DA #1
0934- 00        2740             .DA #0
                2750 *--------------------------------
0935- 03        2760 P.READ      .DA #3
0936- 60        2770 R.DEVNUM    .DA #$60
0937- 00 0C     2780             .DA BUFFER.TWO
0939-           2790 PUSH.VARS .EQ *
0939- 00 00     2800 R.BLOCK     .DA 0
```

```
                2810 *--------------------------------
093B-           2820 BYTES.PER.ENTRY          .BS 1
093C-           2830 ENTRIES.PER.BLOCK        .BS 1
093D-           2840 ACTIVE.FILE.COUNT        .BS 2
093F-           2850 CURRENT.FILE.COUNT       .BS 2
0941-           2860 CURRENT.ENTRY.NUMBER     .BS 1
0942-           2870 NEXT.BLOCK               .BS 2
0B-             2880 PUSH.COUNT               .EQ *-PUSH.VARS
                2890 *--------------------------------
0944-           2900 NEST.LEVEL               .BS 1
                2910 *--------------------------------
BB-             2920 WASTED .EQ *+255/256*256-*
0945-           2930        .BS WASTED
                2940 *--------------------------------
0A00-           2950 BUFFER.ONE  .BS 512
0C00-           2960 BUFFER.TWO  .BS 512
                2970 *--------------------------------
```

Allow BSAVE to New Non-Binary Files in BASIC.SYSTEM 1.1

.....Mark Jackson
Chicago, Illinois

I consider it a bug: BASIC.SYSTEM doesn't allow BSAVEing to a new file unless the type is binary. Yet it is equally desirable to be able to BSAVE to non-binary files without first CREATEing them.

I discovered this problem while implementing FIG-FORTH in ProDOS when I wanted to save the data blocks using as little code as possible. and at the same time allow use of standard text-file word processors.

BSAVEing would solve the code length problem, but to make a text file I would have had to CREATE the file first, thus decreasing speed and increasing code length. Therefore I looked for the BSAVE code inside BASIC.SYSTEM to fix the bug.

As it comes from Apple. BASIC.SYSTEM's parser puts the specified type in $BE6A and then the BSAVE processor places it there again. I used the space this redundant code took for my patch.

There seems to be no good reason for Apple to purposely prevent BSAVEing to new non-binary files, so I think my patch is both worthwhile and safe.

The following applies only to Apple's BASIC.SYSTEM version 1.1. which is the latest as far as I know. The addresses shown are the actual running position. If you want to patch the SYS file by BLOADing at A$2000, then addresses $ADxx will be at $37xx and addresses $AExx will be at $38xx.

The following is in the CREATE code.

Now is:

```
AD41- A9 0F    LDA #$0F     DEFAULT SYS FILE
AD43- 8D 6A BE STA $BE6A    PUT IN GLOBAL PAGE
```

Change to:

```
AD41- A2 0F     LDX #$0F
AD43- 8E 6A BE STX $BE6A
```

The following is in the BSAVE code, and is only reached if it is a new file:

Now is:

```
ADF5- A9 06     LDA #$06      ASSUME TYPE IS BIN
ADF7- 8D 6A BE STA $BE6A      PUT IN GLOBAL PAGE
ADFA- 8D B8 BE STA $BEB8      SET-FILE-INFO LIST
ADFD- AD 56 BE LDA $BE56      CHECK IF TYPE GIVEN
AE00- 29 04     AND #$04
AE02- D0 0E     BNE $AE12     IF YES. THEN ERROR
AE04- 20 46 AD JSR $AD46      CREATE NEW FILE
```

Change to:

```
ADF5- AE 6A BE LDX $BE6A      FILE TYPE FROM PARSING
ADF8- AD 56 BE LDA $BE56      CHECK IF TYPE GIVEN
ADFB- 29 04     AND #$04
ADFD- D0 02     BNE $AE01     IF YES SKIP DEFAULT
ADFF- A2 06     LDX #$06      DEFAULT BIN FILE
AE01- 8E B8 BE STX $BEB8      SET-FILE-INFO LIST
AE04- 20 43 AD JSR $AD43      GO CREATE FILE
```

Thanks to Don Worth and Pieter Lechner for their help in dis-assembling, through their book "Supplement to Beneath Apple ProDOS." (This is the book you get by sending in $10 and a coupon from Beneath Apple ProDOS.)

New Catalog Revisited.....Robert F. O'Brien
Dublin, Ireland

In the May issue of AAL Bob S-C published my article "A New Catalog for Dos 3 3" - he failed to mention or take credit for the fact that he modified my routine and managed to leave a whopping 17 spare bytes - which is 16 more than I left. I was happy enough to have added the new features.

At the end of that article Bob S-C set the challenge to add the Disk Volume message back. However. I have another possible use for those 17 spare bytes - well at least 14 of them!

How about a single-key format control feature for the Catalog command? The user issues the CATALOG command normally; then one more keypress will select either a normal or double-barrelled Catalog display.

Once you install the following additional code, when you issue the CATALOG command the routine waits for a keypress. If you press "D" you get a double-barrelled Catalog listing for your 80-column card or printer Any other keypress will result in the normal 40-column version.

The line numbers on the 14-byte routine which follows make the code fit into the listing from the May article.

```
                1320 CATALOG
AD98- 20 0C FD 1321         JSR MON RDKEY     await keypress
AD9B- 49 8E    1322         EOR #$8E    "D" ($C4) eor LSR ($4A)
AD9D- C9 4A    1323         CMP #$4A    if was "D", now LSR
AD9F- F0 02    1324         BEQ .0      ...it was "D"
ADA1- A9 38    1325         LDA #$38    SEC opcode
ADA3- 8D 21 AE 1326 .0      STA DBL.SWITCH    set option
                .
                .
                .
                2010 DBL.SWITCH SEC
                .

                2150         .BS 3          three free bytes.
```

The code above is of the deadly self-modifying variety, so beware.

Note that if you have version 2.0 of the S-C Macro Assembler, you can write line 1322 as EOR #"D"^$4A if you wish.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada and Mexico; add $14 postage for other countries. Back issues are available for $1 80 each (other countries add $1 per back issue for postage).